Modeling computational algorithms using nonlinear storytelling methods of computer game design

Letonsaari, Mika

Elsevier Scientific Publ. Co
2017-11-01

Letonsaari , M & Selin , J 2017 , Modeling computational algorithms using nonlinear storytelling methods of computer game design . in A Klimova , A Bilyatdinova , J Kortelainen & A Boukhanovsky (eds) , 6th International Young Scientist Conference on Computational Science, YSC 2017, 01-03 November 2017, Kotka, Finland . Procedia Computer Science , Elsevier Scientific Publ. Co , Amsterdam , pp. 131-138 , International Young Scientist Conference on Computational Science , Kotka , Finland , 01/11/2017 . https://doi.org/10.1016/j.procs.2017.11.169

http://hdl.handle.net/10138/232440
https://doi.org/10.1016/j.procs.2017.11.169

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 119 (2017) 131–138

Procedia Computer Science

www.elsevier.com/locate/procedia

6th International Young Scientists Conference in HPC and Simulation, YSC 2017, 1-3 November 2017, Kotka, Finland

Modeling computational algorithms using nonlinear storytelling methods of computer game design

Mika Letonsaari, Jukka Selin

South-Eastern Finland University of Applied Sciences, Patteristonkatu 3 D, 50100 Mikkeli, Finland

Abstract

Computational algorithms can be described in many methods and implemented in many languages. Here we present an approach using storytelling methods of computer game design in modeling some finite-state machine algorithms and applications requiring user interaction. An open source software Twine is used for the task. Interactive nonlinear stories created with Twine are applications that can be executed in a web browser. Storytelling approach provides an easy-to-understand view on computational algorithms allowing communication with people with no computer science education. It also allows rapid prototyping and testing in mixed background work teams.

Peer-review under responsibility of the scientific committee of the 6th International Young Scientist conference in HPC and Simulation

Keywords: Twine; visual programming; rapid prototyping; algorithm design; digital storytelling

1. Introduction

Computer algorithms can be described in many methods such as flowcharts, pseudocode listings, state diagrams, illustrations, and universal modeling language (UML) models. While these models are understandable for those who have education in computer science, they can be too technical and unattractive for laymen.

Here we present an alternative way of presenting computational algorithms and computer programs using storytelling methods of computer game design. The method can also be used as a rapid prototyping tool as the model is an executable program.

Computer programs are sometimes depicted as stories in basic computer science education, especially for children and novices. The reason for this is twofold. Firstly humans are very good in understanding stories. Verbal stories

* Corresponding author. Tel.: +358-44-649-5186.
E-mail address: mika.letonsaari@xamk.fi

1877-0509
Peer-review under responsibility of the scientific committee of the 6th International Young Scientist conference in HPC and Simulation
10.1016/j.procs.2017.11.169

were the way to store and communicate information before the written language. This is made possible by the highly developed episodic memory of human brain.

Secondly, simple programming tasks implemented using procedural programming languages have a sequential structure that resembles much of natural language stories.

The relation of human language and computer programming is also evident from the viewpoint that computer programs are realizations of models in our minds. Alan J. Perlis writes in the forewords of the classic computer science textbook Structure and Interpretation of Computer Programs [1]:

> Every computer program is a model, hatched in the mind, of a real or mental process. These processes, arising from human experience and thought, are huge in number, intricate in detail, and at any time only partially understood. They are modeled to our permanent satisfaction rarely by our computer programs.

And even though computer languages are very formal and their logic is strict in the mathematical sense, there is evidence that human brain understands programming tasks more like a linguistic task than a mathematical reasoning [2].

This close relationship of human language and programming can also be seen from how we speak about programming. For example we casually say that programs are written using language, not coded or created using formal syntax and logical rules of a certain system.

But there are also differences between computer programs and human storytelling. The main difference is that a story tells just a single linear chain of events. Contrary to that, computer algorithms are often characterized by conditional branching of the program execution flow. Traditional story can describe for example a sorting algorithm execution when a single predetermined list is given as an input. Or it can describe a single game of chess.

Whereas a computer program implementing a sorting algorithm can sort any list given to it. And it can use the input values given by players to realize any legal game of chess.

In this paper we will examine the use of storytelling as a method of describing computational algorithm and computer programs to provide an alternative approach to understanding computations.

2. Methods

In this article we study the relationship of computer game design principles and computational algorithms and computer programs. For this work we use an open source license software Twine originally developed by Chris Klimas [3]. Twine has a modern codebase and is developed in Node.js. It can be used in a web browser or as a standalone software.

Interactive nonlinear stories created with Twine are text based games playable in an internet browser. It provides a low learning curve approach into digital storytelling and simple application development. Twine has been used as a storytelling tool with children as young as ten years old [4].

Here we will describe how it compares to some of the relevant programming languages and describe its task domains. We will also describe the tasks used for evaluating the method including the reasoning for the selection requirements.

2.1. Programming language

There is no one correct solution for choosing a modeling language or a programming language. The choice depends on many things such as task domain, performance requirements, library compatibilities, language portability, and developers experience. In this research we aim to present an easy to understand method for understanding computer programs and computational algorithms.

The approach compares closely to many visual languages used in education, such as Scratch and Lego Mindstorms NXT-G, and in engineering and simulation, such as LabView and Simulink.

It is also related to hypertext based programming such as using hypertext markup language HTML with client side Javascript or server side programming such as PHP, Perl, Python or other common web programming languages and frameworks.

The nonlinear story writing software Twine used here is a very versatile software. Nonlinear stories generated with it can be exported in many formats. For example when creating a state of the art 3D game using one of the most popular game development software Unity 3D, Twine stories can be imported using UnityTwine plugin to be used in Unity 3D software [5, 6].

In Twine the stories are written as a hypertext flowchart in a very simple syntax. Links are marked in double brackets and new flowchart nodes are created and named automatically. A control language with many advanced features is used to control the story flow. The control language is somewhat similar to Javascript or PHP in HTML hypertext documents. In this article we use a minimal set of features of the control language, namely variables and conditional branching.

The selection of features is purposefully quite limited to maintain a low abstraction level. Features can seem limited if compared to other languages as most common procedural programming languages have often several sorsts of looping structures (for, while, until, etc.), set of conditional branching of the program execution flow (if.. then.. else), and definition and calling of subroutines or functions.

But these features are of course not always necessary. For example loop instructions do not exist in machine languages of simple central processing units (CPUs) such as the MOS Technology 6502 [7]. Neither they exist as such in functional programming languages or in the fundamental universal Turing machine itself.

2.2. Computational models

Twine is first evaluated in implementing some finite-state machine (FSM) tasks. The finite-state machine is a theoretical model of computation where an abstract machine has a finite number of states and it can be in exactly one state at every moment. The change from one state to another (or back to itself) is called a transition.

FSM have limited computational capacity because it has no memory. In these tasks Twine can be used as a plain hypertext tool without variables and without using any programming control structures. This gives very easy to understand executable models.

Finite-state machines are used in simple automaton such as vending machines, elevators, traffic lights, and combination locks where they work as classified transducers, acceptors, classifiers, and sequencers [8]. Some memory can be simulated by creating more states, but this is not practical for complex applications.

More computationally powerful systems can be created by adding memory to the system. A single memory stack systems called pushdown automata can realize all context-free languages. Random access memory or two memory stacks allow system to simulate universal Turing machine and therefore all possible computer programs.

The standard way of proving Turing completeness of a system is to implement Turing machine or other system already known to be Turing complete with a given system. Turing completeness of Twine is not proved here but can be expected to hold when the default story format Harlowe with arrays, allowing random access memory, is used.[9]

Instead of that, some application examples are presented as an example of how Twine can be used. Twine is most useful in interactive applications and tasks requiring user interaction, such as human computer interfaces, control systems and games.

3. Results

Some sample applications and results in implementing them in Twine are presented here. A turnstile machine is a simple example of a finite-state machine. An implementation of 99 Bottles of Beer song is used for presenting a control structure of a program. Prisoner's Dilemma is used as an example of a slightly more complex problem.

3.1. Finite-state machine models

A turnstile (Figure 1, left) is a simple device which can be modeled as a finite-state machine [10]. At the starting position the gate of the machine is locked. If a coin is inserted into the machine (or keycard shown, or other form of opening signal is given), the gate opens. When the gate is used to enter, it lets one person to pass and then locks again.

The principle of the operation of a turnstile can be described using a state diagram presented in Figure 1 (right). It can be easily seen that this operation can be implemented by using bare hypertext markup language such as HTML as a state machine. For this first example we present HTML language code for comparison.

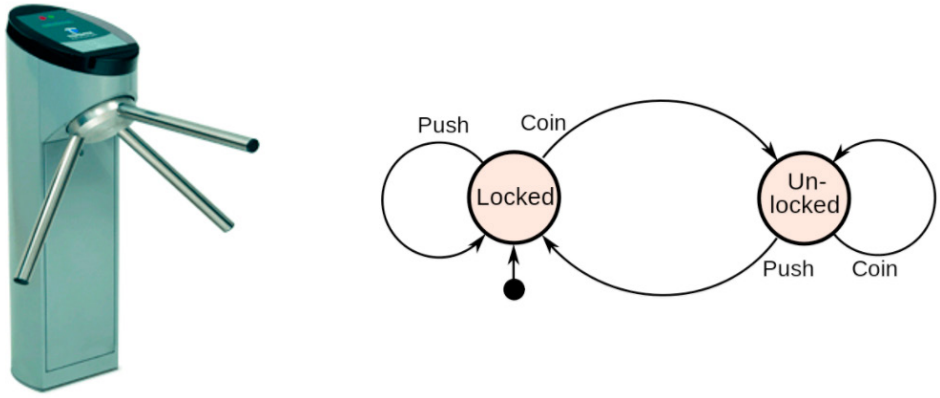


Fig. 1. Left: A turnstile machine (by Wikimedia Commons user Sebasgui, GFDL). Right: State diagram for a turnstile machine (by Wikimedia Commons user Chetvorno, CC0)

Listing 1. closed.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Closed</title>
</head>
<body>
<a href="open.html">Insert a coin</a><br/>
<a href="closed.html">Push</a><br/>
</body>
</html>
```

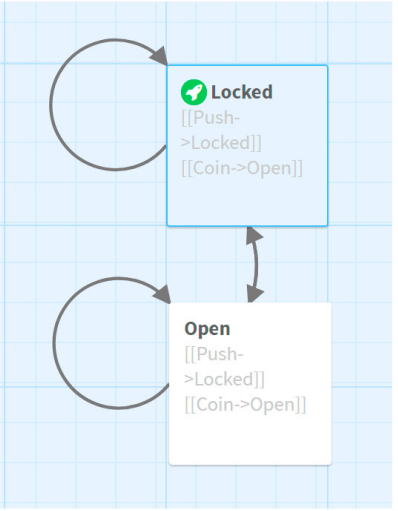
Listing 2. open.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Closed</title>
</head>
<body>
<a href="open.html">Insert a coin</a><br/>
<a href="closed.html">Push</a><br/>
</body>
</html>
```

The turnstile implemented in Twine is presented in Figure 2. Compared to HTML code we can see that the code in HTML is more verbose as it is designed for more general use cases. Especially if we write valid HTML document as described in standards, document type, head section, title, and closing tags are required even if not necessary for providing a result that works in most web browsers.

Twine is thus directed more for this kind of applications. Twine also provides a graphical presentation of the structure of the code or story which makes its use easy in this kind of application.

For a more complex example of using FSM, let us consider parsing for a legal numbers character by character. This task, commonly presented as a railroad diagram, is implemented in Twine in Figure 3. A number can start with a zero, a minus sign or a nonzero digit. For example, if the number starts with a zero, it must end or continue with a decimal dot. After decimal dot there must be at least one digit before the end.



Listing 3: locked

```
[[Push->Locked]]
[[Coin->Open]]
```

Listing 4: open

```
[[Push->Locked]]
[[Coin->Open]]
```

Fig. 2. A turnstile machine implemented in Twine

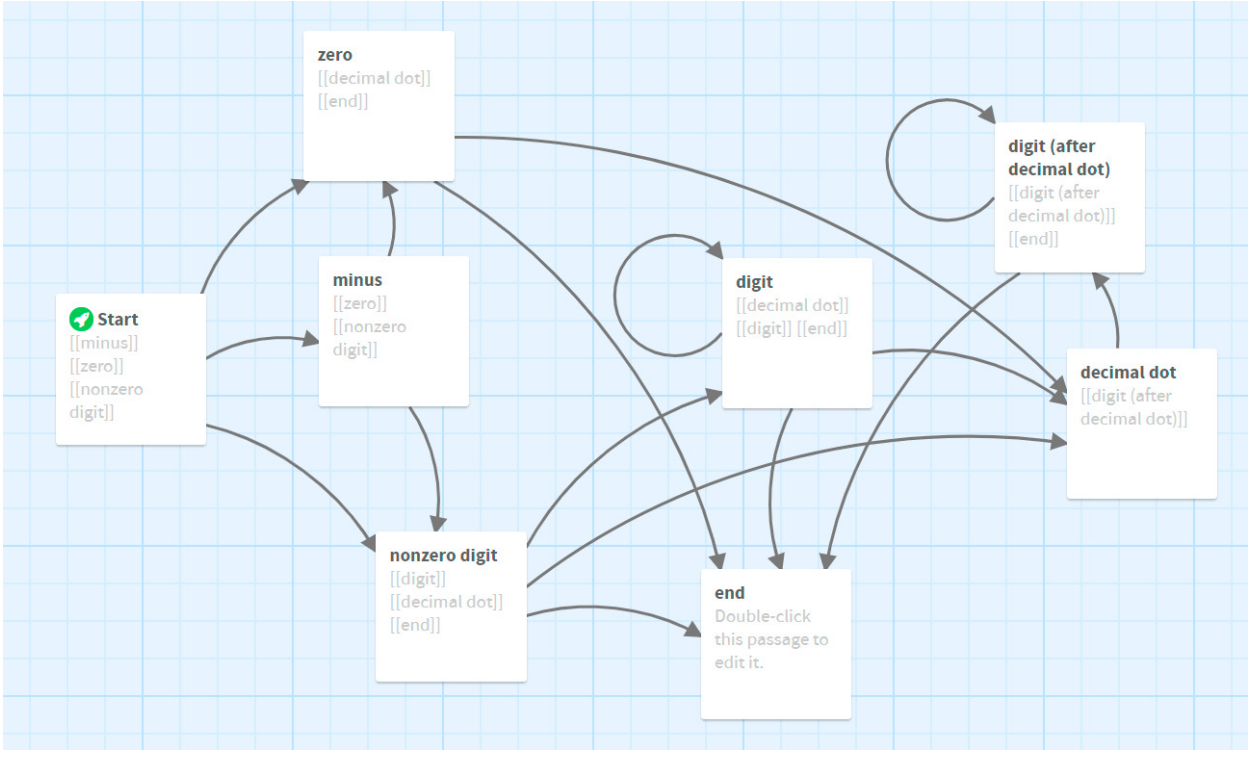
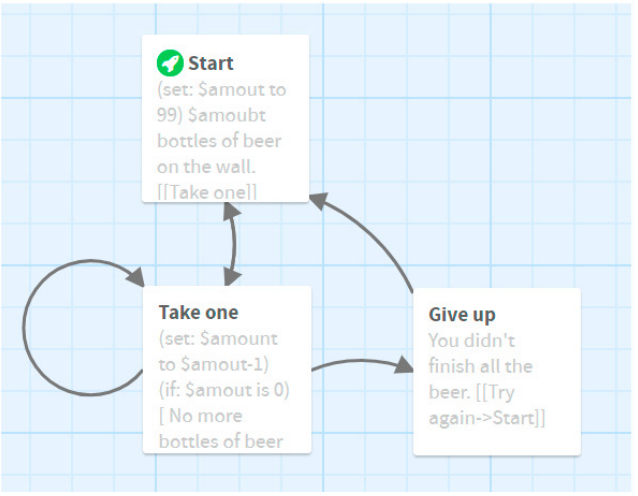


Fig. 3. Parser for legal numbers implemented in Twine

3.2. Example application implementation: 99 Bottles of Beer

As explained earlier, only basic features of the control language of Twine are used in these examples to maintain a low level of abstraction. A classic programming exercise is to implement the 99 Bottles of Beer song. This presents looping backwards from 99 to 0 and it is especially relevant here as we do not use built-in looping structures.



Listing 5: Start

```
(set: $amout to 99)

$amount bottles of beer on the wall.

[[Take one]]

[[Take one]]
```

Listing 6: Take one

```
(set: $amount to $amout-1)

(if: $amout is 0)[
No more bottles of beer on the wall.

[[Go to the store and buy some more.->Start]]
](else:)[
$amount bottles of beer on the wall.

[[Take one]]
[[Give up]]
]
```

Listing 7: Give up

```
You didn't finish all the beer.

[[Try again->Start]]
```

Fig. 4. ”99 Bottles of Beer” implemented as an interactive game in Twine

For a review of the problem, see http://www.99-bottles-of-beer.net/ where the song lyrics are presented in over 1500 programming languages. The complexity aspects of the song in computer programming point of view are discussed by Knuth [11].

We implement an advanced version of the task in Twine where user can control the story and either keep taking bottles or quit the song at will. If the song is aborted, user can start it again. The implementation is shown in Figure 4 and listings 5-7.

3.3. Example application implementation: Prisoner's dilemma

As a final application example, let us consider the famous Prisoner's dilemma of game theory. The Prisoner's Dilemma is very widely studied in the mathematics of economy and many variations of it exist of course. We use here the original formulation of the problem.[12]

Two prisoners, A and B, can either stay silent or betray their partner. If both stay silent, they are given a small penalty, worth -1. If both betray each others, thet are each given bigger penalties worth -2. But if another stays silent and the other betrays, the one who betrays get away without any penalty, and the one who got betrayed get a large penalty worth -3. The payoff matrix is show in Figure 5.

A \ B	B stays silent	B betrays
A stays silent	-1 / -1	0 / -3
A betrays	-3 / 0	-2 / -2

Fig. 5. The payoff matrix of Prisoner's Dilemma

The implementation of the Prisoner's Dilemma in Twine is presented in Figure 6. In graphical programming the code is divided into blocks. Code listings are available at author's website and not presented here in detail.

What we can see from the graphical presentation is that the structure of the program is rather easily seen. From the Begin block there are two arrows, one to Silent and another to Betray. From these blocks there are again two arrows corresponding to the choice of the other player. And then the next round is played starting from the Begin block.

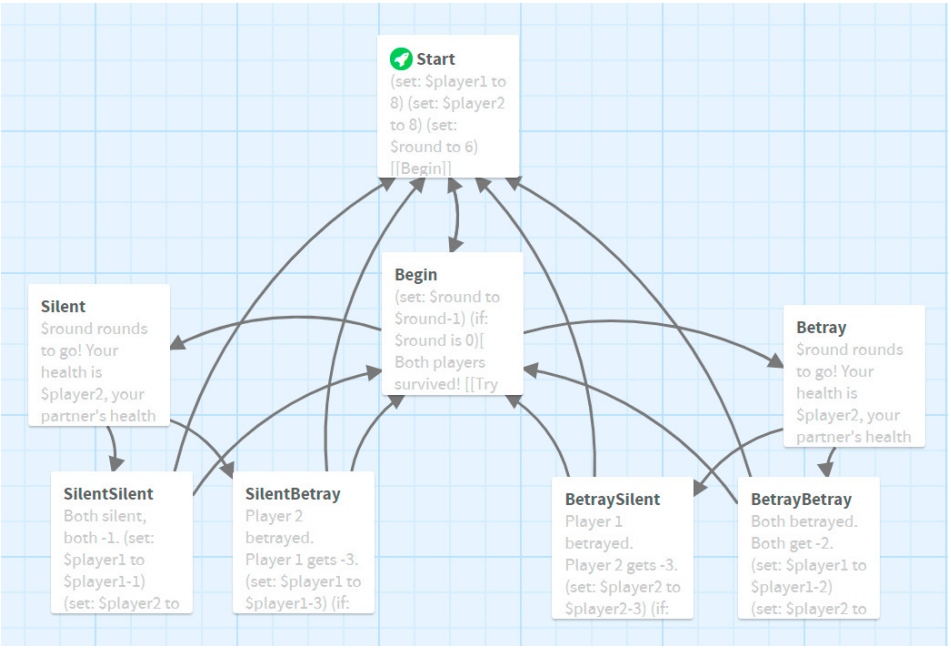


Fig. 6. Twine implementation of Prisoner's Dilemma

Here the graphical presentation is chosen so that it is compact in size and arrows do not cross blocks on their way. The presentation can be chosen by other criteria as well of course. In Figure 7 the structure of the graphical presentation is chosen to resemble the payoff matrix show in Figure 5. This might be a useful property for some application although the complexity of the program doesn't probably justify the use in most cases.

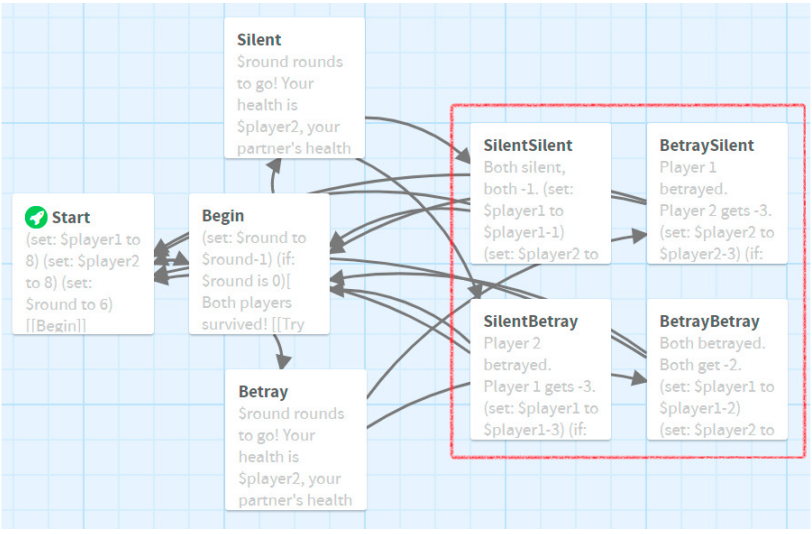


Fig. 7. The structure of the graphical presentation of the program set up to represent the payoff matrix (compare to Figure 5)

4. Conclusions

There is always a need to communicate about the ideas of computer science with people outside the field. For this purpose many simplified and often visual tools are used. In this paper a game design inspired method was presented.

Twine is most natural with simple applications such as finite-state machine implementations or interactive applications with simple logic. While its capabilities can be extended by using more complex features such as Javascript scripting, its main strength is in simplicity.

Martin Fowler, the author of UML Distilled textbook, describes three modes or uses of UML models. Models can be used as a sketch of an idea, where they are used to communicate and work out ideas. Models can be used as blueprints where the main idea is to check that the idea is complete and possible to realize. The third use is to use models as a programming language. This requires model to be complete enough to allow translation to executable code.[13]

These three modes can be recognized in Twine applications as well. The structure of the program can be sketched first. Its completeness can be checked so that there are no dead links in the visual graph. Details needed in the programming can be added in the same sketch and blueprint model. This creates an easy to use work flow.

The storytelling aspect of Twine software is shown in the structure of programs. They resemble web programming where interactive content is produced in the form of hypertext documents. They also resemble early procedural programming such as Basic language where user interface was created with print and input commands and code branching was done with Goto commands [14].

This paper demonstrated basic use of game development software in basic programming and rapid prototyping tasks. Authors have already used the nonlinear storytelling method as a tool for allowing people with very low computer skills to co-operate with computer game development

Future work will include research on using Twine as a tool for multidisciplinary teams to communicate structured ideas and work processes in non-information technology related fields. There will also be research on cognitive and pedagogical aspects of nonlinear storytelling in learning computational thinking.

References

[1] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and interpretation of computer programs*. Justin Kelly, 1996.

[2] Janet Siegmund, Christian Kästner, Sven Apel, Chris Parnin, Anja Bethmann, Thomas Leich, Gunter Saake, and André Brechmann. Understanding understanding source code with functional magnetic resonance imaging. In *Proceedings of the 36th International Conference on Software Engineering*, pages 378–389. ACM, 2014.

[3] Twine is an open-source tool for telling interactive, nonlinear stories. url: http://twinery.org/. Accessed 2017-07-28.

[4] Kelly M Tran. her story was complex: A twine workshop for ten-to twelve-year-old girls. *E-Learning and Digital Media*, 13(5-6):212–226, 2016.

[5] Joonas Korhonen et al. Pelisuunnittelu ja pelin ensimmäisen prototyypin toteuttaminen. 2016.

[6] Mika Letonsaari, Jukka Selin, and Mikko Lampi. Co-creative serious games design process using nonlinear storyline editing. In *Proceedings of the 9th International Conference on Computer Supported Education*, pages 582–588, 2017.

[7] Inc. MOS Technology. 6501-6505 microprocessors, 1975. url: http://archive.6502.org/datasheets/mos_6501-6505-mpu_preliminary_aug_1975.pdf. Accessed: 2017-7-28.

[8] Robert M Keller. Computer science: Abstraction to implementation. *Harvey Mudd College, Claremont, CA, United States*, 2001.

[9] Leon Arnott. Harlowe 2.0.1 manual, 2017. url: https://twine2.neocities.org/. Accessed: 2017-7-28.

[10] David R Wright. Finite state machines. *Carolina State University*, page 203, 2005.

[11] Donald E Knuth. The complexity of songs. *Communications of the ACM*, 27(4):344–346, 1984.

[12] Robert J Aumarm. Acceptable points in general cooperative n-person games. *In Contributions to the Theory of Games (AM-40)*, 4:pp. 287–324., 1959.

[13] Martin Fowler. Uml mode, 2003. url: http://www.martinfowler.com/bliki. Accessed: 2017-7-28.

[14] A Pal. Print 64: a user's guide to the commodore 64, 1988.